به‌نام خداوند بخشنده و مهربان

تمرین سوم: زمان اجرا
درس: تحلیل سیستم داده‌های حجیم

استاد: محمدعلی نعمت‌بخش
دستیاران: فاطمه ابراهیمی، پریسا لطیفی، امیر سرتیپی

نام و نام‌خانوادگی: زلفا شفرئی
آدرس گیت: https://github.com/zolfaShefreie/wordcount_spark_vs_hadoop

- **لطفا پاسخ تمارین حتما در سامانه‌ی کوئرا ارسال شود.**
- **لطفا پاسخ‌های خود را در خود سند سوال نوشته و در قالب یک فایل PDF ارسال کنید.**
- **نام سند ارسالی HW-{homework number}-{Name Family}-{student number}**
- **تمامی فایل‌های مورد نیاز این تمرین در این لینک قابل دسترس است.**
- **خروجی از هر مرحله‌ی تمرین را در سند خود بارگذاری کنید.**

در این تمرین هدف ما مقایسه زمان اجرا در هدوپ و اسپارک است.

برای این منظور ۴ فایل داده متنی با حجم‌های ۱، ۵، ۱۰ و ۱۲ گیگابایتی در اختیار شما قرار گرفته است که انتظار می‌رود با نوشتن برنامه‌ی شمارش کلمات عملیات نگاشت-کاهش را برای داده‌ها بر روی هدوپ و اسپارک انجام دهید. نتایج را گزارش و مقایسه‌ای بین آنها انجام دهید.

آدرس فایل‌ها:

/user/ebrahimi/hw3-data

نمونه‌ی دستور اسپارک را با client mode هم امتحان کرده و تفاوت حالت cluster و client را بیان کنید.

تمامی کدها در گیت‌هاب بارگذاری شده‌اند و با دستور های زیر برروی کلاستر دانشگاه در آدرس /home/zolfa_shefreie/ بارگذاری شده‌اند.

جدول ۱: دستورات استفاده‌شده برای گیت

برای بارگذاری اولیه‌ی مخزن گیت‌هاب	git clone <github_http_address>

git fetch	برای بررسی تغییرات در مخزن
git pull	برای دریافت تغییرات مخزن

چگونگی اجرا برروی هدوپ

کدهای مربوط به اجرای عملیات نگاشت-کاهش در دو فایل mapper.py و reducer.py قرار گرفته است. این کدها با استفاده از پکیج hadoop-streaming.jar برروی هدوپ اجرا می‌شود. مسیر این پکیج در آدرس زیر قرار دارد:

home/hduser/hadoop/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar/

با استفاده از دستور زیر برنامه‌ی نگاشت-کاهش اجرا خواهد شد و تمامی خروجی‌ها با پیشوند hadoop در آدرس /user/zolfa_shefreie/ برروی فایل سیستم هدوپ قرار گرفته‌اند.

hadoop jar {path of hadoop-streaming} -files {path of mapper file},{path of reducer file} -mapper {path of mapper file} -reducer {path of reducer file} -input {path of input file} -output {path of output}

```python
class TextCleaner:
    punc = '''!()-[]{};:'"\,<>/?@#$%^&*_~`|'""_____'''
    # removed_plus = stopwords

    @classmethod
    def run(cls, text: str) -> list:
        """
            run textCleaner for input string
        Args:
            text (str): input string
        Returns:
            list: token list
        """
        return cls.tokenize(cls.normlize(text))

    @classmethod
    def tokenize(cls, text: str) -> list:
        """
            tokenize, delete eng stopwords
        Args:
            text (str): input string
        Returns:
            list: tokens
        """
        # stop_words = cls.removed_plus
        stop_words = list()
        tokens = text.split()
        return [token.strip() for token in tokens if token.strip() and token.strip() not in stop_words]
```

شکل ۱: قسمتی از فایل mapper.py

```python
    @classmethod
    def normlize(cls, text: str) -> str:
        """
            lower, delete numbers, delete punc, strip
        Args:
            text (str): input string
        Returns:
            str: normlized string
        """
        text = text.lower()
        text = re.sub(r"\d+", '', text)

        for each in cls.punc:
            text = text.replace(each, " ")

        text.strip()

        return text


if __name__ == "__main__":
    """
    mapper
    """

    text_cleaner = TextCleaner()

    for line in sys.stdin:
        words = text_cleaner.run(line)
        for word in words:
            print('%s\t%s' % (word, 1))
```

شکل ۲: قسمتی از فایل **mapper.py**

```python
#!/usr/bin/env python
# coding: utf8
import sys

if __name__ == "__main__":

    word_dict = dict()

    for line in sys.stdin:
        line = line.strip()
        word, count = line.split("\t", 1)

        try:
            count = int(count)
        except:
            continue

        word_dict[word] = word_dict.get(word, 0) + count

    word_dict = dict(sorted(word_dict.items(), key=lambda item: item[1], reverse=True))

    for word in word_dict:
        print('%s\t%s' % (word, word_dict[word]))
```

شکل ۳: فایل **reducer.py**

چگونگی اجرا برروی اسپارک

کد عملیات نگاشت-کاهش برروی فایل wordcount.py قرار گرفته است. این فایل برای اجرا دو آرگومان «--hdfs_input» که مسیر فایل ورودی در فایل سیستم هدوپ و «--hdfs_output» که مسیر پوشه‌ی خروجی در فایل سیستم هدوپ می‌باشد. برای اجرا از دستور زیر استفاده می‌شود و تمامی خروجی‌ها با پیشوند spark در آدرس /user/zolfa_shefreie/ قرار می‌گیرند.

spark-submit --master yarn --deploy-mode {mode} {path of wordcount.py} --hdfs_input {path of input} --hdfs_output {path of output}

```python
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
import time
import argparse


def delete_punctuation(x):
    new_str = str() + x
    for each in '''!()-[]{};:'"\,<>/?@#$%^&* ~`|‘“”—–''':
        new_str = new_str.replace(each, "")
    return new_str.strip()


def validate_args(args):
    if not (args.hdfs_input and args.hdfs_output):
        raise Exception("enter path of files")
    else:
        return f"hdfs:{args.hdfs_input}", f"hdfs:{args.hdfs_output}"
```

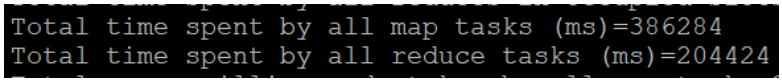
شکل ۴: قسمتی از کد نگاشت-کاهش با استفاده از اسپارک

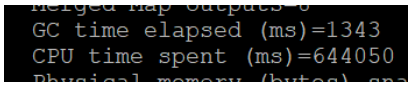
```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--hdfs_input", help="path of input hdfs file")
    parser.add_argument("--hdfs_output", help="path of output hdfs file")
    input_path, output_path = validate_args(parser.parse_args())

    # conf = SparkConf().setAppName("app_sh")
    # sparkContent = SparkContext(conf=conf)
    spark = SparkSession.builder.appName("wordcount_sh").getOrCreate()
    sparkContent = spark.sparkContext

    text_rdd = sparkContent.textFile(input_path)
    start_time = time.time()
    print("start map-reduce actions")
    words = text_rdd.flatMap(lambda line: line.split(" ")).filter(lambda x: x.strip())
    words = words.map(delete_punctuation).map(lambda x: x.lower())
    result = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b).sortBy(lambda a: a[1], ascending=False)
    result.saveAsTextFile(output_path)
    print(f"map-reduce actions done in: {time.time() - start_time}(seconds) ")
```

شکل ۵: قسمتی از کد نگاشت-کاهش با استفاده از اسپارک

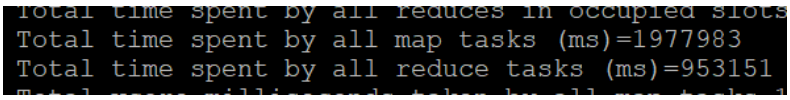
نتایج اجراها

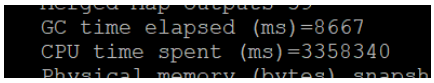
زمان اجرا در برای فایل ۱ گیگ:



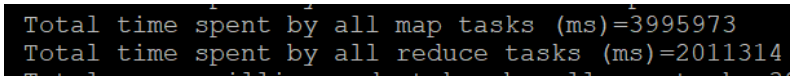


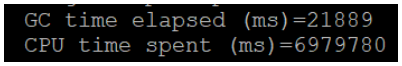
زمان اجرا در برای فایل ۵ گیگ:



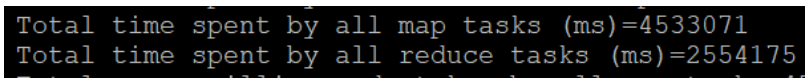


زمان اجرا در برای فایل ۱۰ گیگ:

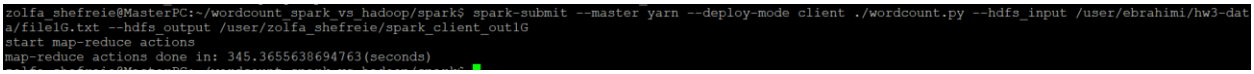


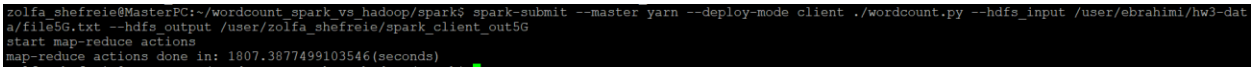


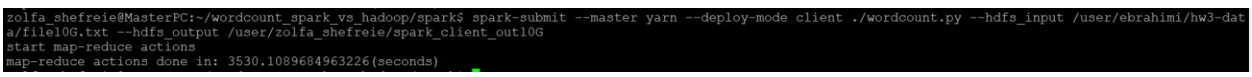
زمان اجرا در برای فایل ۱۲ گیگ:



زمان اجراها به‌ترتیب برای فایل‌های ۱، ۵، ۱۰ و ۱۲ گیگ:







```
zolfa_shefreie@MasterPC:~/wordcount_spark_vs_hadoop/spark$ spark-submit --master yarn --deploy-mode client ./wordcount.py --hdfs_input /user/ebrahimi/hw3-dat
a/file12G.txt --hdfs_output /user/zolfa_shefreie/spark_client_out12G
start map-reduce actions
map-reduce actions done in: 4067.2643995285034(seconds)
zolfa_shefreie@MasterPC:~/wordcount_spark_vs_hadoop/spark$
```

جدول ۲: مقایسه‌ی زمان اجرای هدوپ و اسپارک

file	Runtime of Hadoop(s)	Runtime of spark client mode(s)
File1G	644.05	345.36
File5G	3358.34	1007.38
File10G	6979.78	3530.11
File12G	7326.32	4067.26

طبق مقایسه‌های انجام‌شده زمان اجرای اسپارک برای اجرای برنامه‌ی نگاشت-کاهش بسیار بهینه‌تر از هدوپ می‌باشد.

تفاوت حالت cluster و client

در حالت cluster درایور اسپارک در یک فرایند اصلی برنامه اجرا می‌شود که توسط yarn برروی خوشه مدیریت می‌شود و مشتری می‌تواند پس شروع برنامه حذف شود یا کار را فراموش کند. در حالت client درایور در فرایند درخواست‌کننده اجرا می‌شود و برنامه‌ی master فقط برای درخواست منابع از yarn استفاده می‌کند بنابراین درخواست‌کننده باید آنلاین باشد و با خوشه در تماس باشد. بنابراین برای حالت‌هایی که درخواست‌ها از دور انجام می‌شود منطقی‌ترین حالت، استفاده از حالت cluster می‌باشد [۱].

از تفاوت‌هایی که در اجرای این دو حالت برروی کلاستر دانشگاه حس شد می‌توان به موارد زیر اشاره کرد:

- در برنامه‌ی wordcount.py دستوراتی برای چاپ و لاگ‌گرفتن موجود بود. در حالت client این دستورات اجرا می‌شد اما در حالت cluster این نتیجه‌ی این دستورات نشان داده نمی‌شد که علت آن طبق پاراگراف بالا مشخص است.
- زمان اجرای در حالت cluster بسیار طولانی‌تر از حالت client بود که اجرا برای فایل ۱ گیگ به ۳ ساعت می‌رسید. به‌دلیل چاپ نشدن لاگ‌ها امکان درک شروع برنامه وجود نداشت پس قاطعانه نمی‌توان گفت زمان اجرای طولانی

برای زیاد بودن زمان اختصاص دادن کار به یک فرایند در خوشه‌ی اسپارک به‌دلیل ازدحام بود یا زمان اجرای عملیات نگاشت-کاهش.

مرجع

[1] "Cluster vs Client: Execution modes for a Spark application," 19 May 2020. [Online]. Available: https://blog.knoldus.com/cluster-vs-client-execution-modes-for-a-spark-application/.